Lessons Learned from CMR Cloud Prototype

Goal

Document lessons learned from our CMR Cloud Prototyping activity.

Traceability

CMR Cloud Phase 1 Task

Prototyping Summary

In the process of setting up tests of the primary goal of the prototype to evaluate different solutions for Oracle, Elasticsearch, and RabbitMQ we gained valuable experience utilizing the AWS environment. We learned how best to run our applications, ways to automate setup in AWS, as well as approaches and techniques for taking advantage of AWS that would not be possible on our own infrastructure. For our recommendations for the primary goal of the prototype effort see Recommendations from CMR Cloud Prototype.

AWS Services

S3

- Scaled to 10K writes a second after talking to AWS team and having them manually partition our bucket.
- Need to make sure we work with our rep to get workload and operations bucket manually partitioned.
- Reads scale to about 1K second when using 50 threads.
- 99% of single file retrievals in 250ms.
- Deleting buckets requires manually deleting everything in the bucket. AWS commandline has a --force option that will delete everything.
    - Takes a really long time to delete lots of objects - we're going to have to write some code.
- There's no good way to get a count of how many items are in an S3 bucket.
- Intermittent failures happen when writing and retrieving (697 Exceptions when populating ~300 million records - roughly 2 errors per million requests).
    - Note: found out afterwards many of these failures happened at a time when Amazon was in the process of manually partitioning our S3 bucket. True error rate may be much lower.
- Seems like a perfect use case for the data centers' archives (great for large files).
- Latency cannot compete with databases. For small files databases continue to make sense.

EC2

- Enable termination protection on instances that you plan to keep around for a while.
- Hit limits pretty easily for things like spinning up too many instances or exceeding EBS limits
- Spinning up lots of instances with a script is really easy
- Power to scale up and down is great. Things like one time populations can be done really quickly this way.
- Testing out various scenarios is cheap and easy
    - Is this workflow CPU bound?
    - If I use a machine with twice as many cores do I see performance double?
    - How much memory do I actually need to run my application
    - At the same cost is performance of my application better with lots of smaller nodes or fewer more powerful nodes?

EBS

- EBS optimized is important to set on EC2 instances
    - Bootstrap was 3 times slower without EBS optimized set on the VMs
- General Purpose SSD Performance is much better than expected
    - Able to use General Purpose SSD for both Elasticsearch cluster and for Oracle
    - Did not find a need to use provisioned IOPS for anything (much more expensive)

Oracle RDS

- There were many options for loading data into Oracle, but they all required a direct connection to a database
    - New database migration service seemed promising to keep two systems in sync
        - Found it did not migrate sequences
        - Poor performance - migrated less than 10% of ops holdings in a day.

        - Eventually created a database link from workload to RDS, copied ops database dump files to workload, used the Orace file transfer utility to transfer the dump files, then loaded the database dump files into Oracle RDS
- General SSD storage (no provisioned IOPS) seems fine
- Able to handle 10K granule reads per second sustained for hours during the population of S3
- A couple of client SQL connection exceptions during loading of S3
- Make sure we configure the parameters the same way we do Oracle on premise (things like the optimizer mode of 11g, talk to Bob for a full list)
- Supports sharing database snapshots across accounts
    - Useful for bringing up an ops backup in a prototyping / sandbox environment

Dynamo

SNS / SQS

- Getting the list of subscribers to a topic has a high variance in terms of the time it takes. This should be avoided on a per-publish basis - better to cache the results or simply hard code the topics if they are not changing.

AMIs

- Packer can be used to build AMIs, but its generally easier to start a base instance with an existing AMI and configure it by hand and then snapshot it to create an AMI.
- Creating instance store AMIs is much more complicated than creating EBS backed AMIs.
- For prototyping we found it better to create simple base AMIs and then configure them / install packages on startup using user data. This allowed us to easily try different configs for Elasticsearch, etc. It also allowed us to start up multiple Elasticsearch clusters without them trying to join a cluster hard-coded into the AMI.

APIs / Console / SDK

- Prefer using the command line API in lieu of the AWS console for starting instances.
- Multiple ways to interact with AWS - Java API, Clojure API, Console, Python command line
- Java SDK turned out to be much faster than Clojure library (which used reflection) for reading and writing to S3
    - Requests went from 25ms down to 10ms

Elasticsearch S3 Snapshots

- Really easy to setup and run
- Snapshot failed due to 500 error from S3 on one of the copies the first time - changed configuration of snapshot repository to retry up to 5 times on failures
- Full snapshot took 44 minutes
- Subsequent snapshots perform deltas
- Were able to bring up a new cluster without any data and then use restore to populate with operational data
- Restore took approximately 90 minutes to get to yellow state and 3 hours to get to green state
- This will make it really easy to create temporary workload environments that have real data.

Load Balancers

- Really easy to configure
- Have features we need
    - Add instances to load balancers
    - Request timeout
    - Health check endpoint
    - How many consecutive failures before taking a node out
    - How many consecutive successes before putting a node back in
- Set the idle timeout to whatever the maximum expected response time is for a call to that API. Default of 60 seconds was too low for us with elasticsearch.

IAM

- Allows you to create roles that have permissions to access services at different levels (Read only, Full access, etc.). These roles can then be assigned to users or instances. This allows you to bring up instances that can access your RDS, SNS, etc., without needing to set environment variables for AWS credentials.

AWS Support

- Tickets to increase limits are generally handled in hours.
- Helped with S3 bucket performance. Took a couple of tries to get the right information conveyed, but then resolved within a day.

AWS Observations

- AWS services are being added frequently. Database migration service was added after we started the prototyping effort.
- Seem to be a lot of options to do the same thing.
- Have not seen issues with node stability
    - Only issue was when we spun up 100 nodes and one did not come up successfully
- AWS services seem to work really well. We should look to use services whenever possible rather than maintaining our own EC2 instances.
- AWS network is lower latency than our network. Some elasticsearch requests take just 2 ms to return back to the search app.
    - Make sure when choosing/building AMIs that you build in advanced networking support (10GB/s).
- Cloud provides a good solution for handling Elasticsearch upgrades and monthly security patches.

Approaches and Techniques

- Prototyping and Testing
    - Cloud makes testing scenarios, architecture changes, and configuration changes really simple (key takeaway)
        - Elasticsearch examples
            - Storage
            - Shard configuration
            - Number of nodes
            - Additional replica impact to query and index performance
            - Memory configuration impacts
        - We can setup environments quickly even with operational data
            - Start up a new Oracle RDS instance from operational database backup
            - Spin up an elasticsearch cluster and restore from operational S3 snapshot
- Script everything
    - Much faster to do things in bulk using a commandline script
    - Used scripts for starting up Elasticsearch clusters and App servers
- parallel-rsync and parallel-ssh have been useful for performing the same task on many instances
- Centralized logging is needed and should be figured out from the start
    - Create an Elasticsearch/Kibana/Logstash (ELK) stack early to centralize logs and make them searchable.

Miscellaneous

- Siege was an excellent fit for our load testing for the prototype
    - Simple to setup - just apt-get install siege
    - Simple to create a driver file and run
    - Flexible enough to handle all of our load testing scenarios for the prototype
        - Oracle RDS + S3
        - Elasticsearch
        - Amazon SNS/SQS
    - Should look into whether we can use it for our workload testing going forward
        - Does not support running queries based on timestamps which is something we thought would be useful for replaying ops queries
- Able to get CMR applications running without many changes
    - search
    - metadata-db
    - cubby
    - index-set
    - indexer
    - bootstrap